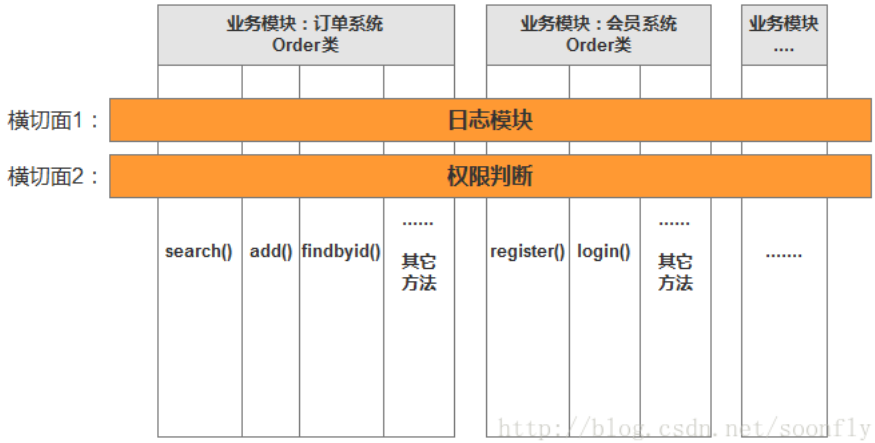
【Spring学习30】Spring AOP：基于XML配置和注解实现

作者：soonfly (/authorarticle.html?author=soonfly) 2017-04-13 ☆ 收录到我的专题 (/select.html?articleId=401737)

标签 aop (http://www.bijishequ.com/info/search.html?searchText=aop) 开火 (http://www.bijishequ.com/info/search.html?searchText=开火) 方法 (http://www.bijishequ.com/info/search.html?searchText=方法) 切面 (http://www.bijishequ.com/info/search.html?searchText=切面) twmspringaopdemo (http://www.bijishequ.com/info/search.html?searchText=twmspringaopdemo)

我们知道Spring以IoC(Inverse of Control 反转控制)和AOP(Aspect Oriented Programming 面向切面编程)为内核。
AOP（Aspect Oriented Programming），即面向切面编程，是OOP（Object Oriented Programming，面向对象编程）的补充和完善。
举个栗子（用的伪代码），假设我们要在系统的每个方法被调用时，用logger.log()记录方法开始运行的时间。于是不得不在每个业务方法里加上logger.log()这行代码。在这个场景中，logger.log()与业务无关，而且散布的到处都有。这种散布在各处的无关的代码被称为**横切（cross cutting）**。而且今后进行优化，想把方法结束时间也记录下来，就不得不跑到每个方法的末尾将logger.log()再加一遍。

这显然不符合程序员“懒惰”的天性。因此AOP这种思想出现了，AOP技术利用一种称为”横切”的思路，再利用动态代理技术，把统一的与业务无关的工作代码比如刚才的logger.log()，动态织入到各个方法中去，以达到减少系统的重复代码，降低模块之间的耦合度，增加可维护性的目的。



AOP编程很简单，拿刚刚这个栗子来说，程序员只要做三件事：
1、写原有业务组件代码。比如订单Order类包含的search(),add(),findbyid()，或者是User类包含的register(),login()。
2、写要切入的功能代码。如例子中的日志记录功能： `Class OSSHelp(){public void log(){ logger.info(.....) } }`
3、定义切面(Aspect)。

第一、二两件事很好理解，而且本来就是该做的。关于第三点定义切面(Aspect)是干嘛呢？
其实也很简单，切面(Aspect)就是要说明3W，即**what,where,when**：
what（做什么）：要做什么呢，当然是让每个方法都执行test.log()方法了。
where（在哪做）：术语叫**切入点（pointcut）**。就是我要让哪些方法执行test.log()呢？是要在所有方法中都执行，还是只在Order类的方法中执行？
when（什么时侯做）：是在方法调用前执行test.log()还是在方法调用后执行呢？共有五种：前置、后置、异常、最终、环绕（前置+后置+异常+最终）。

有了上面的基础，现在来看看AOP中的几个常用术语：
切入点(Pointcut) = where（在哪做）：例如某个类或方法的名称，可以用正则表达式来指定。
通知(Advice) = what（做什么）+ when（什么时侯做）
切面(Aspect) = 通知(Advice) + 切入点(Pointcut)

连接点(joinpoint)：切入点的类型，如：方法，字段，构造函数。Spring只支持方法类型的连接点，所以在Spring中连接点指的就是被拦截到的方法。
织入(Weaving)：将切面应用到目标对象并创建代理对象的过程。
织入有三种时机：
1、运行时：切面在运行时被织入,SpringAOP就是以这种方式织入切面的，原理是使用了JDK的动态代理或CGLIB代理。
2、编译时：当一个类文件字节码被编译时进行织入，这需要特殊的编译器才可以做的到，例如AspectJ的织入编译器。
3、类加载时：使用特殊的ClassLoader在目标类被加载到程序之前，改变目标类，增强类的字节代码。

说了很多，还是举栗子说明吧。现在我们要来开发一款星际战争的游戏。

首先写一个接口叫Fireable,这是一个牛X的接口,能对一切对象造成伤害:

```
package twm.spring.aopdemo;
public interface Fireable {
int attack(Object obj);
}
```

然后写一个Tank(坦克)类，它实现了开火接口：

```
package twm.spring.aopdemo;
public class Tank implements Fireable{
@Override
public int attack(Object obj) {
System.out.println("坦克开火！造成100点伤害！");
return 100;
}
}
```

星际战争怎么能缺少飞机，因此再实现一个FighterPlane（战斗机）类：

```
package twm.spring.aopdemo;
public class FighterPlane implements Fireable{
@Override
public int attack(Object obj) {
System.out.println("战斗机开火！造成200点伤害！");
return 200;
}
}
```

在Spring配置文件中注册：

```
<bean id="tank" class="twm.spring.aopdemo.Tank" />
<bean id="fighterPlane" class="twm.spring.aopdemo.FighterPlane" />
```

调用：

```
public static void main(String[] args) throws Exception {

Object tempTarget = new Object();

ApplicationContext ctx = new ClassPathXmlApplicationContext("beans.xml");
Fireable fighterPlane = ctx.getBean("fighterPlane", Fireable.class);
Fireable tank = ctx.getBean("tank", Fireable.class);
fighterPlane.attack(tempTarget);
System.out.println();
tank.attack(tempTarget);

}
```

输出：

> 战斗机开火！造成200点伤害！
>
> 坦克开火！造成100点伤害！

主业务开发完成，而且运行的很不错。
不久，新的需求来了。它要求：**攻击前要记录开火时间，攻击完成后向指挥部报告：完成攻击**。
普通青年觉得这没什么，在每一个类的attack()方法中添加记录开火时间和报告完成的代码不就行了。嗯，这样确实可以，现在只有两个实现类：飞机和坦克，因此只要添加两次就行了。但是随着业务的发展，后面还有更多能开火的类加入，比如航母、迫击炮、激光台、离子炮塔，整个系统中可能多达成百上千种实现，一个个去加的话，就成了2B青年了。

现在AOP正式登场

在编码之前先下载两个包：aopalliance.jar，aspectjweaver.jar，并引入工程。Maven的话请添加好依赖。

一、基于XML配置Aop

先为新的需求添加一个实现类：

```
public class FireAssist {
/*记录开火时间*/
public void ActionLog() throws Throwable {
System.out.println("开火时间："
+ (new SimpleDateFormat("yyyy-MM-dd HH:mm:ss"))
.format(new Date()));
}
/*报告已完成开火*/
public void ReportComplete() throws Throwable {
System.out.println("报告长官：打完收工！");
}
}
```

然后到Spring配置文件中进行配置：

```
<bean id="tank" class="twm.spring.aopdemo.Tank" />
<bean id="fighterPlane" class="twm.spring.aopdemo.FighterPlane" />
<!-- 下面是新添加的 -->
<bean id="fireAssist" class="twm.spring.aopdemo.FireAssist" />
<!-- Aop根元素 -->
<aop:config>
<!-- 切面(Aspect) -->
<aop:aspect ref="fireAssist">
<!-- 切点 -->
<aop:pointcut expression="execution(* twm.spring.aopdemo.*.*(..))" id="pc1"/>

<!-- 通知(Advice) -->
<aop:before method="ActionLog" pointcut-ref="pc1"/>
<aop:after method="ReportComplete" pointcut-ref="pc1" />
<!-- 通知也可这样写 <aop:before method="ActionLog" pointcut="execution(* twm.spring.aopdemo.*.*(..))"/> -->
</aop:aspect>

<!-- 可加多个切面(Aspect) -->

</aop:config>
```

其它什么都不变，再运行代码，输出：

```
开火时间：2017-04-13 20:51:07
战斗机开火！造成200点伤害！
报告长官：打完收工！

开火时间：2017-04-13 20:51:07
坦克开火！造成100点伤害！
报告长官：打完收工！
```

切面配置说明

可以看到通过 `<aop:config />` 元素，就将fireAssist内的两个方法织入到所有的attack()方法中了。

`<aop:config>` 是进行AOP设置的顶级配置元素，类似于这种东西。

`<aop:aspect>` 定义一个切面，下面有这些子元素：

`<aop:after>` 后通知

`<aop:after-returning>` 返回后通知

`<aop:after-throwing>` 抛出后通知

`<aop:around>` 周围通知

`<aop:before>` 前通知

`<aop:pointcut>` 定义一个切点

定义切点的表达式

execution(* twm.spring.aopdemo.* . *(..))

这样写代表twm.spring.aopdemo包下所有的类的所有方法。

第一个*代表所有的返回值类型

第二个*代表所有的类

第三个*代表类所有方法

最后一个..代表所有的参数。

任意公共方法执行：

execution(public * *(..))

任何一个名字以”attack”结尾的方法：

execution(* *attack(..))

任何一个名字以”attack”开头的方法：
execution(* attack*(..))

实现Fireable接口的类的任意方法：
execution(* twm.spring.aopdemo.Fireable.*(..))

twm.spring.aopdemo包下所有的类的所有方法:
execution(* twm.spring.aopdemo.* .*(..))

在twm.spring.aopdemo包下的任意连接点,不包括子包：
在spring下，连接点只能是方法，也就是twm.spring.aopdemo包下的所有类的所有方法：
with(twm.spring.aopdemo.*)

在twm.spring.aopdemo包下的任意连接点,包括子包：
with(twm.spring.aopdemo..*)

二、使用注解配置AOP

即然使用注解，那么先把Spring配置文件中的内容全删了。
接下来开始：
第一步：用注解方式将Fireable的实现类注册到容器
为业务类Tank和FighterPlane添加注解：

```
@Component
public class Tank implements Fireable{
@Override
public int attack(Object obj) {
System.out.println("坦克开火！造成100点伤害！");
return 100;
}
}
```

```
@Component
public class FighterPlane implements Fireable{
@Override
public int attack(Object obj) {
System.out.println("战斗机开火！造成200点伤害！");
return 200;
}
}
```

第二步：通过注解为FireAssist类配置横切逻辑

```
@Component
@Aspect
public class FireAssist {
/*记录开火时间*/
@Before("execution(* *.attack(..))")
public void ActionLog() throws Throwable {
System.out.println("开火时间："
+ (new SimpleDateFormat("yyyy-MM-dd HH:mm:ss"))
.format(new Date()));
}
/*报告已完成开火*/
@After("execution(* *.attack(..))")
public void ReportComplete() throws Throwable {
System.out.println("报告长官：打完收工！");
}
}
```

@Aspect声明该类是一个切面；@Before表示方法为前置before通知，@After表示后置After通知，通过参数execution声明一个切点。

第三步：配置自动扫描

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:context="http://www.springframework.org/schema/context" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:c="http://www.springframework.org/schema/c" xmlns:aop="http://www.springframework.org/schema/aop" xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.2.xsd http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.2.xsd">

<context:component-scan base-package="twm.spring.aopdemo" />
<aop:aspectj-autoproxy />
</beans>
```

<aop:aspectj-autoproxy /> 标签是让Spring框架自动为bean创建代理。
该标签有一个属性proxy-target-class，如果设置为true，则表明要代理的类是没有实现任何接口的，这时spring会选择Cglib创建代理。讲到这里就应该讲一讲java创建代理的方法：
1、使用Java动态代理来创建，用到InvocationHandler和Proxy，该方式只能为接口实例创建代理。
2、使用CGLIB代理，就可以不局限于只能是实现了接口的类实例了。
spring aop首先选择Java动态代理来创建，如果发现代理对象没有实现任何接口，就会改用cglib。刚这儿说到的proxy-target-class，如果设置为true，就是强制使用cglib创建代理。

调用：

```
public static void main(String[] args) throws Exception {

Object tempTarget = new Object();

ApplicationContext ctx = new ClassPathXmlApplicationContext("beans.xml");
Fireable fighterPlane = ctx.getBean("fighterPlane", Fireable.class);
Fireable tank = ctx.getBean("tank", Fireable.class);
fighterPlane.attack(tempTarget);
System.out.println();
tank.attack(tempTarget);

}
```

输出：

```
开火时间：2017-04-13 20:57:25
战斗机开火！造成200点伤害！
报告长官：打完收工！

开火时间：2017-04-13 20:57:25
坦克开火！造成100点伤害！
报告长官：打完收工！
```

打完收工！如果需要更深入，就去查文档。

原文链接：http://blog.csdn.net/soonfly/article/details/70162442